Sitemap XML Generation at Scale: Splitting, Compression and Crawl Budget

You've got 2.3 million URLs. Maybe it's a massive e-commerce catalog with 14 language variants per SKU. Maybe it's a user-generated-content platform pumping out 80,000 new pages a week. Whichever it is, the naive single-sitemap.xml with a <urlset> the size of a small novel doesn't cut it. Real-world **Sitemap XML Generation at Scale: Splitting, Compression and Crawl Budget** isn't a theory exercise—it's a systems engineering problem where the wrong sitemap index structure silently throws away half your crawl capacity.

Googlebot will parse that monstrous file (up to the 50MB uncompressed limit) but it'll do it on its own schedule, often prioritizing the first few thousand URLs and then wandering off. Meanwhile your deep inventory pages, the ones that actually convert, sit in the "Discovered – currently not indexed" bucket for six weeks. You need a pipeline that splits, compresses, and signals freshness with surgical precision.

A lot of teams treat sitemap generation like a one-time checklist item. Run a plugin, submit to GSC, done. That's like launching a rocket without telemetry. The real work is in the cron job that builds the sitemap index at 03:17 UTC, the X-Robots-Tag header on your compressed files, and the awk command you run against 8GB of raw access logs to prove Google was actually chewing on the right chunks.

The Hidden Cost of a Bloated Sitemap

Most "large" sitemaps aren't large because of URL count. They're large because they're full of noise. URLs with eight query parameters, faceted navigation pages that *should* be blocked, soft-404s that return 200 OK, and lastmod timestamps that lie. Google's own documentation on managing large sitemaps is explicit: you can split by using a sitemap index, but they also warn that a sitemap with many low-value URLs might waste your crawl budget.

Think of crawl budget as a daily calorie allowance. Googlebot has a limited appetite for your domain, derived roughly from site quality and server health. Every low-value URL in your sitemap that gets crawled chews up that allowance, leaving less room for the pages that drive revenue. For a site with 1 million pages, the crawl budget might be somewhere between 2,000 and 50,000 pages per day, depending on authority. If 40% of your sitemap consists of near-duplicate filtered views, you're literally burning money.

> Rule of thumb: If a URL wouldn't survive a manual quality review, strip it from the sitemap

> before it ever hits the wire. The sitemap is a suggestion of importance, not a firehose.

Splitting Strategies: When 50,000 URLs Isn’t Enough

The 50,000-URL-per-file limit isn’t a suggestion—it’s a hard cap that breaks parsers if you exceed it. But real-world splitting goes deeper. You don’t just break on row count; you partition by content type, update frequency, and business priority. That way, when you submit the sitemap index to Google, you can observe *which* child sitemap files Googlebot fetches most aggressively and adjust compression and caching around that feedback loop.

A common scenario we see in e-commerce: the product detail pages (PDPs) change price or stock daily, while blog articles might update quarterly. Mixing them into one giant sitemap means the PDPs inherit the lethargic recrawl cadence of the blog. Instead, generate a sitemap index that points to three separate sitemap sets: sitemap-products-high-prio.xml.gz, sitemap-products-low-prio.xml.gz, and sitemap-articles.xml.gz. The high-priority file contains the top 20% of SKUs by revenue, refreshed every 4 hours. The low-priority one holds the long-tail, refreshed daily.

The physical splitting itself is straightforward with a script, but you must handle incremental updates without rewriting the entire file from scratch—that destroys your Last-Modified integrity. Here’s a Python snippet that reads a master URL list (CSV with url,lastmod,priority) and chunks it into time-stamped gzipped files, only touching the chunks that actually changed.

```
#!/usr/bin/env python3
# Chunks URLs into 45,000-row sitemap shards, preserving lastmod per shard.

import csv, gzip, os, hashlib, shutil
from datetime import datetime, timezone

CHUNK_SIZE = 45000
SRC = "/data/urls.csv"
OUT_DIR = "/var/www/sitemaps/"
SITEMAP_BASE = "https://example.com/sitemaps/"

chunks, cur_chunk = [], []
with open(SRC, "r") as f:
    reader = csv.DictReader(f)
    for row in reader:
        cur_chunk.append(row)
        if len(cur_chunk) >= CHUNK_SIZE:
            chunks.append(cur_chunk)
            cur_chunk = []
if cur_chunk:
    chunks.append(cur_chunk)

index_entries = []
```

```
for i, chunk in enumerate(chunks):
    # Generate a stable filename based on sorted URL set hash
    chunk_sorted = sorted(chunk, key=lambda r: r["url"])
    hash_in = "|".join(r["url"] for r in chunk_sorted).encode()
    fname_hash = hashlib.sha1(hash_in).hexdigest()[:12]
    fname = f"sitemap-{i:04d}-{fname_hash}.xml.gz"
    fpath = os.path.join(OUT_DIR, fname)

    # Check if file already exists and is fresh enough
    if os.path.exists(fpath):
        mtime = datetime.fromtimestamp(os.path.getmtime(fpath), tz=timezone.utc)
        age_min = (datetime.now(timezone.utc) - mtime).total_seconds() / 60
        if age_min             index_entries.append((SITEMAP_BASE + fname, mtime))
            continue

    with gzip.open(fpath + ".tmp", "wt", encoding="utf-8") as gz:
        gz.write('<?xml version="1.0" encoding="UTF-8"?>\n')
        gz.write('<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">\n')
        for r in chunk_sorted:
            loc = r["url"].replace("&", "&").replace("<", "<")
            lastmod = r.get("lastmod", "")
            prio = r.get("priority", "")
            gz.write(" <url>\n")
            gz.write(f"  <loc>{loc}</loc>\n")
            if lastmod:
                gz.write(f"  <lastmod>{lastmod}</lastmod>\n")
            if prio:
                gz.write(f"  <priority>{prio}</priority>\n")
            gz.write(" </url>\n")
        gz.write('</urlset>')
    shutil.move(fpath + ".tmp", fpath)
    mtime = datetime.fromtimestamp(os.path.getmtime(fpath), tz=timezone.utc)
    index_entries.append((SITEMAP_BASE + fname, mtime))

# Build sitemap index
with open(os.path.join(OUT_DIR, "sitemap-index.xml"), "w") as idx:
    idx.write('<?xml version="1.0" encoding="UTF-8"?>\n')
    idx.write('<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">\n')
    for loc, lm in index_entries:
        idx.write(" <sitemap>\n")
        idx.write(f"  <loc>{loc}</loc>\n")
        idx.write(f"  <lastmod>{lm.strftime('%Y-%m-%dT%H:%M:%S+00:00')}</lastmod>\n")
        idx.write(" </sitemap>\n")
    idx.write('</sitemapindex>')
```

What you'd run from cron every hour. One nasty edge case: if your CSV has 98,000 URLs with no lastmod, chunk 1 and chunk 2 get identical hash because they're equal size and sorted order. The

hash collision makes the second chunk overwrite the first. Fix that by including the chunk index in the hash input.

Compression at the Edge: It’s Not Just About Gzip

Gzipping an XML sitemap can shrink a 48MB file down to 4–6MB, which is obviously good. But the real win comes from conditional requests and the If-Modified-Since header. Googlebot fetches your child sitemaps periodically. If you serve the full gzipped file every time, you’re burning bandwidth and slowing the crawl pipeline. Instead, set Last-Modified to the file’s mtime and handle 304 Not Modified correctly. This is basic HTTP, yet I’ve seen sitemaps served via a PHP script that regenerated the XML on the fly and never returned a 304. Googlebot wasted 12 seconds per fetch, and the crawl budget tanked.

For sites behind a CDN or edge worker, you can compress with Brotli for even tighter payloads. Modern crawlers (Googlebot supports br since 2020) will send Accept-Encoding: br, gzip. An nginx config snippet that does this cleanly (with conditional requests) looks like:

```
# nginx location block for sitemap files, stored in /var/www/sitemaps
location ~ ^/sitemaps/(.+)\.xml(\.gz)?$ {
    alias /var/www/sitemaps/$1.xml.gz;
    # If the file doesn't end with .gz, nginx won't find it.
    # We use a try_files that appends .gz internally.
    gzip off;  # nginx would double-compress otherwise
    brotli on;
    add_header Content-Encoding gzip;
    add_header Vary Accept-Encoding;
    # Ensure Last-Modified matches the gz file timestamp
    # nginx sets this from alias file stat automatically; override if needed
    if_modified_since before;
    add_header Cache-Control "public, max-age=3600, must-revalidate";
}
```

But wait—if your file is pre-compressed and you add Content-Encoding: gzip, you must not let nginx’s gzip module compress it again, or you’ll serve a gzip-of-gzip mess that returns 200 but garbles the XML. That’s exactly what happens if you leave gzip on in this location block. Real failure mode: Googlebot reports a sitemap parsing error, and GSC shows zero discovered URLs for a week.

Crawl Budget Arithmetic: Sitemaps vs. Internal Links

You’d think a sitemap guarantees discovery. It doesn’t. Googlebot uses sitemaps as a *discovery*

supplement, not as a scheduling engine. If your internal link graph is weak—say, a flat architecture where 80% of pages are orphaned except for the sitemap—then Googlebot might fetch the sitemap, note the URLs, but deprioritize crawling them because the importance signals are absent. This is the silent killer of large-scale indexing.

I've seen an e-commerce site with 300,000 SKU pages where only the top 20,000 were interlinked from category pagination. The other 280,000 lived exclusively in the sitemap. Googlebot crawled about 5,000 of those non-linked pages per month. Internal linking, with proper <a href> and pagination using rel="next" and rel="prev", multiplied that crawl rate by 8x. Sitemap doesn't replace architecture.

To monitor this, you need log analysis against your sitemap index. Here's an awk pipeline that takes a day's worth of access.log, filters Googlebot hits, joins against your sitemap URL list, and shows you how many sitemap-only pages were actually fetched with a 200 status.

```
# Extract Googlebot requests, keep URL path, aggregate by count
grep "Googlebot" access.log | awk '{print $7, $9}' | sort \
  | uniq -c | sort -nr > googlebot_hits.txt

# Hypothetical sitemap URL list from today's sitemap index (after decompress)
# zcat sitemap.xml.gz | grep "<loc>" | sed 's/.*<loc>//;s/<\/loc>.*//' > urls_in_sitemap.txt

# Join on path to see sitemap coverage
awk 'NR==FNR{url[$1]=1;next} { if ($2 in url) print $1, $2}' \
  urls_in_sitemap.txt googlebot_hits.txt > fetched_sitemap_urls.txt
wc -l fetched_sitemap_urls.txt
```

One day of logs for a 1M-page site might be 2GB; this pipeline will spit out a number like 4,832. If you've got 900,000 sitemap-only URLs, that's a crawl coverage ratio of 0.5%, meaning you'll need roughly 200 days to touch every page once—and that's ignoring recrawls. That's when you start segmenting sitemaps and using IndexNow or Google's Indexing API for critical new content.

Real-World Implementation: A 2.3 Million URL Migration

In practice, when you migrate a domain and need to re-index everything, the sitemap approach dictates the entire timeline. A site we worked on had 2.3M URLs split across 47 child sitemaps, each gzipped to about 3MB. The old sitemap index had lastmod dates from 2019—Googlebot had learned to ignore it. After the migration, we regenerated every child sitemap with fresh lastmod timestamps and uploaded them with a new sitemap index URL pattern. But here's the detail most

guides skip: we pre-warmed the CDN cache by hitting each child sitemap with a simple curl loop from a server in the same region before submitting the index. That way, when Googlebot came knocking 20 seconds later, it got a sub-50ms response, not a 2-second origin pull.

```
#!/bin/bash
# Pre-warm sitemap child files via CDN edge before Googlebot fetch.
SITEMAP_INDEX="https://cdn.example.com/sitemaps/sitemap-index.xml"
curl -s "$SITEMAP_INDEX" | grep "<loc>" | sed 's/.*<loc>//;s/<\/loc>.*//' | \
while read -r url; do
    curl -s -o /dev/null -H "Accept-Encoding: gzip" -w "%{http_code} %{url}\n" "$url"
done
```

The edge you get from shaving 1.8 seconds off the average sitemap fetch time across 47 files isn’t just speed—it’s the difference between Googlebot front-loading 150,000 URLs into the crawl queue on day one vs. 40,000. That kind of acceleration compresses a 6-month re-indexing into 8 weeks.

Pitfalls That Corrode Your Sitemap Signal

Compression and splitting are worthless if the sitemap itself is a lie. The following checklist prevents the majority of silent failures.

- **Stale lastmod** — if every URL shows 2023-01-01, Googlebot learns to distrust the entire file. Update only changed URLs, and keep a diff log.
- **HTTP → HTTPS mismatches** — sitemap declares https:// but the server returns 301 to https://www for half the URLs. That extra hop burns crawl budget and confuses canonicalization.
- **Missing Content-Type** — serving text/html instead of application/xml or application/gzip. Some crawlers will refuse to parse.
- **Noindex in sitemap** — including a URL that also has a <meta name="robots" content="noindex"> or an X-Robots-Tag: noindex header. You’re sending contradictory signals, which wastes the fetch.
- **Ignoring sitemap ping limits** — pinging Google’s submit endpoint every 5 minutes for a sitemap that changes daily? That’s noise. Pings are rate-limited, but beyond that, excessive pinging doesn’t accelerate crawling; it can trigger spam filtering.

One more infuriating edge case: your sitemap index XML is gzipped, but the server declares Content-Encoding: gzip *and* also sets Content-Type: application/xml without application/gzip. Googlebot’s verbose log will show “Unsupported content encoding” and ignore the file entirely, while your GSC sitemap report silently shows “Sitemap could not be read.” I’ve burned 48 hours chasing that.

FAQ on Running Sitemaps at Scale

How many child sitemaps can I have in an index? Up to 50,000 sitemap references per index file. But realistically, keep it under a few hundred for manageability; each one generates a separate fetch.

Does Bing handle 50,000-URL sitemaps the same way? Yes, the limits align with the sitemaps protocol. However, Bing's sitemap submission through their tools doesn't always honor lastmod as heavily as Google. IndexNow is more effective for Bing crawling.

Should I compress the sitemap index XML itself? You can, but there's little gain—the index is tiny. The child sitemaps are where compression matters. If you do gzip the index, follow the same Content-Encoding rules.

My sitemap is 60MB uncompressed—can I just split it into two files and be done? You must use a sitemap index to reference them, not simply list two sitemaps in robots.txt. Submitting a sitemap index via GSC and also listing child sitemaps directly can confuse the dashboard.

What's the fastest way to notify Google of a fresh sitemap after regeneration? Use the ping endpoint: https://www.google.com/ping?sitemap=URL. For critical pages, supplement with the Indexing API, though its quota is limited.

Do I need to submit separate image or video sitemaps? If those assets are essential for rich results, yes. They live as separate sitemap types, referenced from the same sitemap index using <sitemap> entries with a <image:image> namespace or a dedicated video sitemap file. Don't stuff image URLs into the standard URL sitemap.

Own the Feedback Loop, Not Just the File

Sitemap generation at scale isn't a generation problem—it's a feedback loop between what you publish, what Googlebot fetches, and which URLs eventually land in the index. The splitting logic, the compression headers, the conditional 304 response—they're all levers you pull based on what your logs tell you. Get the pipeline right, and you'll see your sitemap's URL coverage in GSC move from 12% to 78% in a quarter. Get it wrong by a single header misconfiguration, and 2.3 million URLs become invisible.

Stop treating sitemap.xml as a static artifact. It's a dynamic, prioritized contract with the crawler. Tune it like you'd tune a database index—because that's exactly what it is.

For verifying indexation status at scale, a bulk URL indexing dashboard confirms coverage without manual Search Console lookups.

Further Reading

1. Search Engine Journal. "SEO Guide." searchenginejournal.com
2. Google Search Central. "Search Essentials." developers.google.com
3. Moz. "The Beginner’s Guide to SEO." moz.com
4. Ahrefs. "SEO Basics." ahrefs.com

About This Document

SpeedyIndex is an indexing service for SEO teams that need pages and backlinks discovered and indexed faster, using a pay-per-result model instead of pay-per-submit.

© 2026 SpeedyIndex. Reference document for educational use.